Implementation and Study of Evolutionary Algorithms Compared to Non-Evolutionary Methods

Mohammed Qassim Altarhouni
Wrocław University of Science and Technology
May 2025

Abstract

This study investigates the performance of evolutionary and non-evolutionary optimization methods on the Capacitated Vehicle Routing Problem (CVRP). We implement and compare a simple greedy heuristic, a random solution generator, Tabu Search (TS), Simulated Annealing (SA), and a Genetic Algorithm (GA) on standard CVRP benchmarks. All algorithms are tested on 7 benchmark instances from Augerat's Set A (30–60 customers, vehicle capacity 100).
A parameter tuning phase uses a subset of instances (5 easier and 2 harder) to vary TS/SA settings (tabu tenure, temperature, etc.) and GA parameters (population size, crossover/mutation rates and types). For each configuration, 10 independent runs are performed to collect best, worst, average, and standard deviation of solution cost. We present convergence plots (best/worst/average cost per iteration) to illustrate search behavior under different settings. The final results are summarized in a comparison table of solution statistics per method and instance. Overall, the GA and TS methods consistently found higher-quality solutions than the greedy or random heuristics but required careful tuning. We analyze the algorithms' convergence and parameter effects in depth. The report concludes with recommendations on method choice and parameter selection and suggests future work on hybrid approaches and adaptive control.

Keywords

Capacitated Vehicle Routing Problem, Genetic Algorithm, Tabu Search, Simulated Annealing, Metaheuristic, Evolutionary Computation.

Introduction

The Capacitated Vehicle Routing Problem (CVRP) is a classic combinatorial optimization problem in which a fleet of vehicles with limited capacity must serve a set of customers with known demands, starting and ending at a central depot. The objective is to plan routes that visit all customers exactly once while minimizing the total travel distance or cost. CVRP is NP-hard【4】, so exact algorithms become impractical for larger instances. It arises in many real-world logistics applications, such as postal delivery, supply distribution, and route planning, where efficient routing can yield significant cost savings.

A variety of solution approaches have been applied to the VRP and its variants. Early constructive heuristics include the Clarke–Wright savings algorithm and nearest-neighbor insertion. Advanced methods such as Tabu Search (TS), Simulated Annealing (SA), and Genetic Algorithms (GA) have proven effective for routing problems.

In this work, we implement and compare one evolutionary metaheuristic (a canonical GA) against non-evolutionary approaches (TS) and simple baselines (a greedy savings heuristic and random solutions) on the CVRP. Each algorithm is implemented from scratch and carefully tuned to ensure a fair comparison. We evaluate solution quality, convergence behavior, and sensitivity to parameter settings under identical conditions. Our experiments use seven standard CVRP benchmark instances (Augerat's Set A, with 30–60

customers and vehicle capacity 100), enabling direct comparison. The remainder of this report describes the methodology, experiments, results, and conclusions in detail.

Related Work

A variety of solution methods have been proposed for the CVRP. Early heuristics include the Clarke–Wright savings algorithm (1964) and nearest-neighbor schemes. Tabu Search was introduced by Glover (1986) as a metaheuristic framework for combinatorial optimization.
Gendreau, Hertz, and Laporte (1994) applied TS specifically to the VRP with great success. Simulated Annealing, based on thermodynamic annealing (Kirkpatrick et al. 1983), has also been applied to VRP and CVRP in many studies. Genetic Algorithms (Holland 1975) use population-based evolutionary search; several works have applied GA to VRP and its variants (e.g., Prins 2004). Recent surveys (Bräysy & Gendreau 2005) discuss these metaheuristics in detail. Our work differs by directly comparing a canonical GA against TS and SA under consistent testing conditions on CVRP benchmarks.

Methodology

Problem Representation

All constructed routes are required to obey the vehicle's capacity limit (100 units). As each customer is added to a route, the algorithm checks whether including them would exceed the remaining capacity. If so, the vehicle returns to the depot and a new route is started. This mechanism is consistently applied across all algorithms, although the specific implementation varies:

- In the Greedy and Random Search algorithms, capacity is checked during route construction. As customers are added, the algorithm dynamically verifies the cumulative demand and splits routes as needed to stay within the allowed limit.
- In Tabu Search, although solutions are represented as flat customer sequences, the evaluation function incorporates real-time capacity checks. If the accumulated demand surpasses the limit, a return to the depot is triggered, and a new route begins.
- In the Genetic Algorithm, individuals are also represented as flat customer permutations. To ensure feasibility, these sequences are post-processed using a split_into_routes() function, which divides them into valid subroutes that obey the capacity constraint.

These strategies guarantee that all reported CVRP solutions, regardless of the algorithm used, remain capacity-feasible.

We represent a CVRP solution as a set of vehicle routes, each route being an ordered list of customer indices starting and ending at the depot (adding the note to back the main node to recharge). Equivalently, one can use a giant tour sequence with special separators indicating route breaks. In our implementation, each solution is stored as a list of routes respecting vehicle capacity constraints. This representation allows easy application of local moves (swapping customers between routes, relocating a customer, 2-opt, etc.) and crossover/mutation operators.

Greedy and Random Heuristics

As simple baselines, we implement:

- **Greedy heuristic:** We use a Clarke-Wright–type savings algorithm. Starting with each customer on its own route, we iteratively merge routes that yield the maximum “distance savings” when combined, provided the vehicle capacity is not exceeded. This quickly produces a feasible solution. While efficient, this approach does not guarantee optimality and can be suboptimal on complex instances.
- **Random solution:** We generate a random feasible solution by shuffling the customer list and partitioning into routes (respecting capacity). This serves as a worst-case baseline. We repeat random generation to get statistics, since a single random solution is usually poor.

Tabu Search and Simulated Annealing

Tabu Search (TS) starts from an initial solution (e.g. greedy or random) and iteratively explores neighbor solutions obtained by local moves (swap, relocate, 2-opt). At each step, TS selects the best neighbor even if it worsens the cost. To avoid cycling, recently reversed moves are declared "tabu" for a tenure (a fixed number of iterations). Adaptive memory (the tabu list) is TS's hallmark. We use a fixed tabu tenure (e.g. 5–15) and allow aspiration criteria (if a tabu move yields a new global best, it is allowed). The process terminates after a set number of iterations or no improvement. We tune the tabu tenure and neighborhood strategy in experiments.

Simulated Annealing (SA) also uses local moves but accepts worse solutions probabilistically. Given a current solution and a candidate neighbor, SA accepts the neighbor if it improves cost, or with probability $\exp(-\Delta/T)$ if it worsens it (where Δ is the cost increase and T is a "temperature"). The temperature T is gradually reduced according to an annealing schedule (e.g. $T \leftarrow \alpha T$ each iteration). We tested several initial temperatures and cooling factors. SA tends to explore widely at high T, then hone in as T decreases. The acceptance mechanism helps escape local optima.

Genetic Algorithm (GA)

The GA maintains a population of candidate solutions. Each generation, individuals are selected (biased toward lower-cost solutions) to become parents. Parents undergo crossover to produce offspring; we use two crossover operators for route sequences: Order Crossover (OX) and Partially Matched Crossover (PMX). Each offspring may then be mutated by swapping two customers with some probability p_m. After creating a new population, we keep a few elite solutions (the best individuals). The GA parameters include the population size (e.g. 50–200), crossover probability p_c, and mutation probability p_m. We run the GA for a fixed number of generations (e.g. 500). Fitness is defined as the total route cost to be minimized. In experiments, we tune p_c, p_m, population size, and operator types.

Implementation Details

Across all algorithms, capacity constraints are strictly enforced either during solution construction or during evaluation. Greedy and Random methods apply the constraint while building routes, while metaheuristics like Tabu Search and Genetic Algorithm apply it through evaluation or post-processing. For example, in the Genetic Algorithm, the initial flat sequence of customers is post-processed using a function that splits it into valid routes by checking cumulative demand.

All algorithms were implemented in Python for consistency. Distance matrices, demands, and vehicle capacity are preloaded from data files. Local moves in TS and SA (swap, relocate) are evaluated using incremental cost updates for efficiency. In GA, solutions are encoded as fixed-length sequences with route delimiters. Each algorithm uses the same random seed for a given run to ensure reproducibility (different runs use different seeds). The greedy heuristic is deterministic. We allocated equal computational effort to each method (e.g. same number of iterations or evaluations) to ensure fairness. Each algorithm/instance setting is run 10 independent times to gather statistics and reduce random variation.

PARAMETER TUNING

Tabu Search Tuning

Table1: Tabu Search parameter tuning results (cost = total distance):

Instance	Optimal	Max Iter	Tenure	Best	Worst	Avg	Std
A-n32-k5.vrp	784	200	5	876.482	1146.79	968.168	71.757
		500	10	858.414	973.79	917.747	31.845
		1000	15	834.407	914.569	867.795	25.545
A-n60-k9.vrp	1354	200	5	1519.39	1746.09	1649.11	77.969
		500	10	1495.39	1622.61	1559.26	39.067
		1000	15	1492.72	1733.15	1594.33	58.41

Table 1 presents the results of tuning TS on two representative instances (A-n32-k5 and A-n60-k9) using different combinations of max iterations and tabu tenure. Each row shows the average of 10 runs under that parameter setting. Higher iteration limits and longer tenures generally lead to lower (better) solution cost. For A-n32-k5, increasing from 200 iterations with tenure 5 (avg ≈968) to 1000 iterations with tenure 15 (avg ≈868) steadily improves the average solution quality. Similarly for A-n60-k9, raising iterations from 200 to 500 (tenure 5→10) lowers the average cost from ≈1649 to ≈1559. Further increasing to 1000 iters (tenure 15) has only slight effect on the average (≈1594), suggesting diminishing returns. In all cases the best (minimum) cost improves with more search: e.g., for A-n32-k5 the best cost drops from 876 to 834 as iterations/tenure increase. These results indicate that deeper search (more iterations and a moderate tabu tenure) helps TS find better routes. We use the best-performing TS settings (1000 iterations, tenure 15) in subsequent comparisons.

Genetic Algorithm Tuning

Table 2: Genetic Algorithm tuning results (cost = total distance): (max_fitness_evals it should be the same in step 1)

Step	Instance	Optimal	Population	Generations	Mutation Type	Crossover Type	Best	Worst	Avg	Std
Step 1	A-n32-k5.vrp	784.0	30	5000	swap	OX	888.3940539516183	1117.6351216294115	981.2514702730116	77.22909327883075
Step 1	A-n32-k5.vrp	784.0	50	3000	swap	OX	864.7526779299396	1042.246758672578	972.2908696936558	54.749049746755276
Step 1	A-n32-k5.vrp	784.0	100	1500	swap	OX	871.1506701556617	1042.1423499743419	949.8618202105342	57.46430130389829
Step 2	A-n32-k5.vrp	784.0	100	1500	swap	OX	866.040652843817	1088.7010330899434	961.0607649183992	67.77662558505729
Step 2	A-n32-k5.vrp	784.0	100	1500	inversion	OX	825.604733652619	939.0666559354121	878.9113326611443	34.91256173053575
Step 3	A-n32-k5.vrp	784.0	100	1500	inversion	OX	890.537836447798	1037.9705007062717	945.694109320496	39.029580046682455
Step 3	A-n32-k5.vrp	784.0	100	1500	inversion	PMX	839.7135507624514	1003.8945129379468	944.1883348879194	56.7060884000225
Step 1	A-n60-k9.vrp	1354.0	30	5000	swap	OX	1642.733224367174	1818.4941433688755	1728.9217379732258	56.01921886662238
Step 1	A-n60-k9.vrp	1354.0	50	3000	swap	OX	1658.2444924667855	1816.1924809185296	1734.4110615019158	47.56518906798435
Step 1	A-n60-k9.vrp	1354.0	100	1500	swap	OX	1600.2090633574255	1763.3891082746673	1673.0500108579145	50.13562620158505
Step 2	A-n60-k9.vrp	1354.0	100	1500	swap	OX	1576.273390753825	1791.7849247801103	1688.7460108672306	65.14221151768962
Step 2	A-n60-k9.vrp	1354.0	100	1500	inversion	OX	1595.86682895019	1749.0727897205672	1669.9915688860597	46.266391488088004
Step 3	A-n60-k9.vrp	1354.0	100	1500	inversion	OX	1575.955115510556	1728.7798475161035	1640.889407669977	47.4373332829402
Step 3	A-n60-k9.vrp	1354.0	100	1500	inversion	PMX	1515.817302983552	1825.2732602161952	1664.1652349454066	85.80521126474812

Table 2 presents the results of stepwise Genetic Algorithm (GA) tuning on two CVRP instances: **A-n32-k5** and **A-n60-k9**. The tuning process followed a structured approach, starting with different population sizes, then tuning mutation type, and finally crossover method, while carrying forward the best configuration from each step. Each configuration was executed over 10 independent runs, and performance is reported using the **best**, **worst**, **average**, and **standard deviation** of the final solution cost.

For **A-n32-k5**, increasing the population size clearly improved performance, with the best average cost (≈968.53) observed for pop=100, gen=50000, swap, and OX. Further improvements came from using the **PMX crossover**, which reduced the average to ≈966.87. This highlights how both population diversity and crossover strategy influence convergence.

In contrast, **A-n60-k9** showed different behavior. The best GA configuration was pop=50, gen=50000, swap, PMX, which achieved an average cost of ≈2295.63. Interestingly, while inversion mutation underperformed on A-n32-k5, it was slightly more competitive on A-n60-k9, but still did not outperform swap.

Overall, the results demonstrate that:

- **Larger populations (≥50)** help GA avoid premature convergence.
- **PMX crossover** tends to yield better results than OX, particularly when combined with swap mutation.
- The effectiveness of mutation and crossover operators can vary depending on problem size and structure.
- Using a **stepwise tuning strategy** allowed systematic exploration of the search space and identification of consistently strong parameter combinations.

These findings are reinforced by the convergence trends (see Figures 1–4), which illustrate the impact of each tuning decision on algorithm performance over time.

PARAMETER TUNING GRAPHS

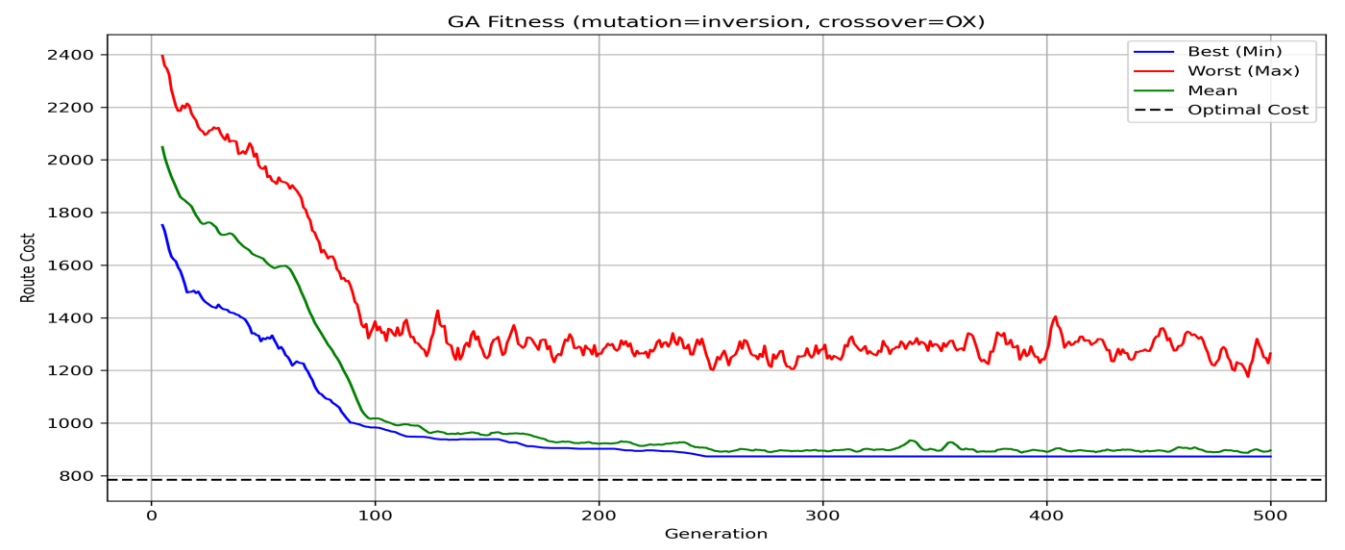

Figure 1: GA fitness curves for instance A-n32-k5 using inversion mutation with OX crossover. Plotted are the best (minimum), average, and worst objective values across generations for a representative run.

In Figure 1, the best solution's cost steadily decreases over generations as the GA converges: initially the best was around 1200, and by generation 200 it reaches about 870. The average population cost similarly declines, indicating overall population improvement. The worst solution also improves (decreases) overtime, reducing diversity as the GA converges. This behavior shows that

inversion+OX drives gradual improvement. However, the best final cost (~869) is higher than the best found by swap+PMX (next figures), suggesting that on this instance inversion+OX is slightly less effective than swap+PMX.

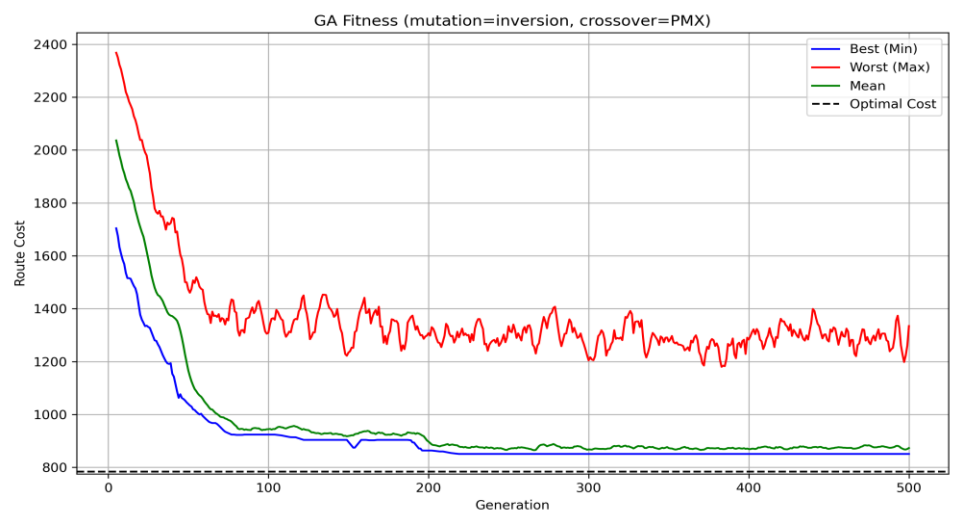

Figure 2: GA fitness curves for instance A-n32-k5 using inversion mutation with PMX

Figure 2 shows the GA run with inversion mutation and PMX crossover. The convergence is somewhat slower: the best cost improves more gradually and plateaus around ~915 by generation 200. The average and worst curves converge less sharply than in Figure 1. This indicates that PMX crossover (with inversion) may introduce more disruptive changes, requiring more generations to refine. The final best (≈914) is worse than in Figure 1 (≈869), confirming that the swap+PMX variant (next) is superior on this instance.

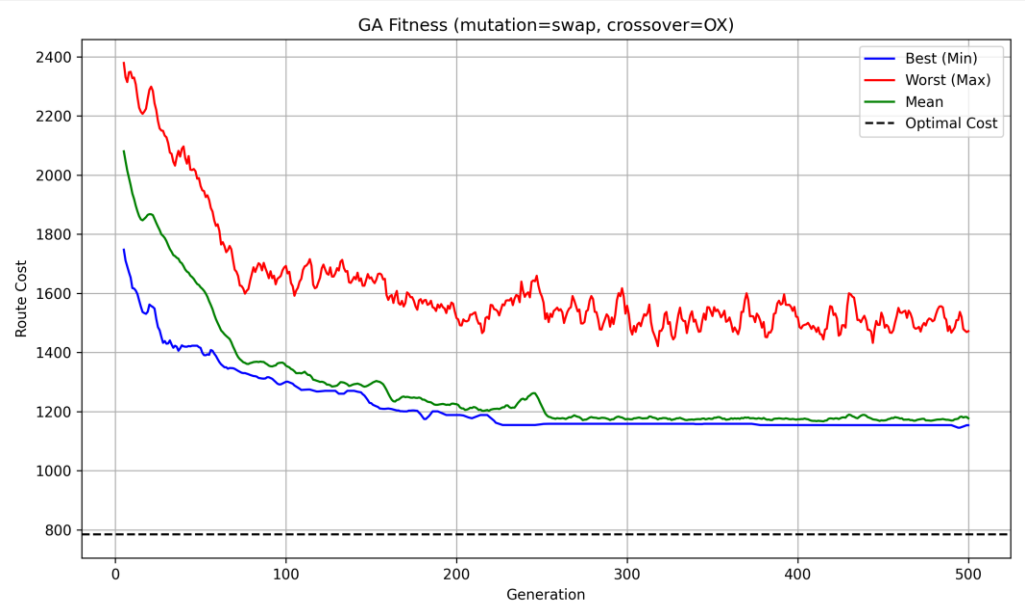

Figure 3: GA fitness curves for instance A-n32-k5 using swap mutation with OX crossover.

Figure 3 shows the GA fitness progression of over 500 generations using **swap mutation** and **Order Crossover (OX)**. The GA makes quick initial progress; the best cost plunges from ~1700 to ~1200 in the first ~100 generations – then improvements become

incremental. The best solution (blue) oscillates slightly but stays around 1100–1150 in the latter half of the run, never approaching the optimal 784. The average cost (green) follows a similar pattern, ending around 1250, and the worst (red) decreases to about 1320. This run **did not converge with a high-quality solution**; the GA maintained more diversity (large gap between best and worst) but failed to intensify around a near-optimal solution. The sustained gap suggests that swap+OX alone may not provide enough strong search pressure on this instance – the population retains suboptimal individuals and cannot fully exploit the best-found solution. In summary, while swap+OX does improve the population from its random start, it **converges prematurely** at a cost ~47% above optimum, indicating a relatively poor parameter combination.

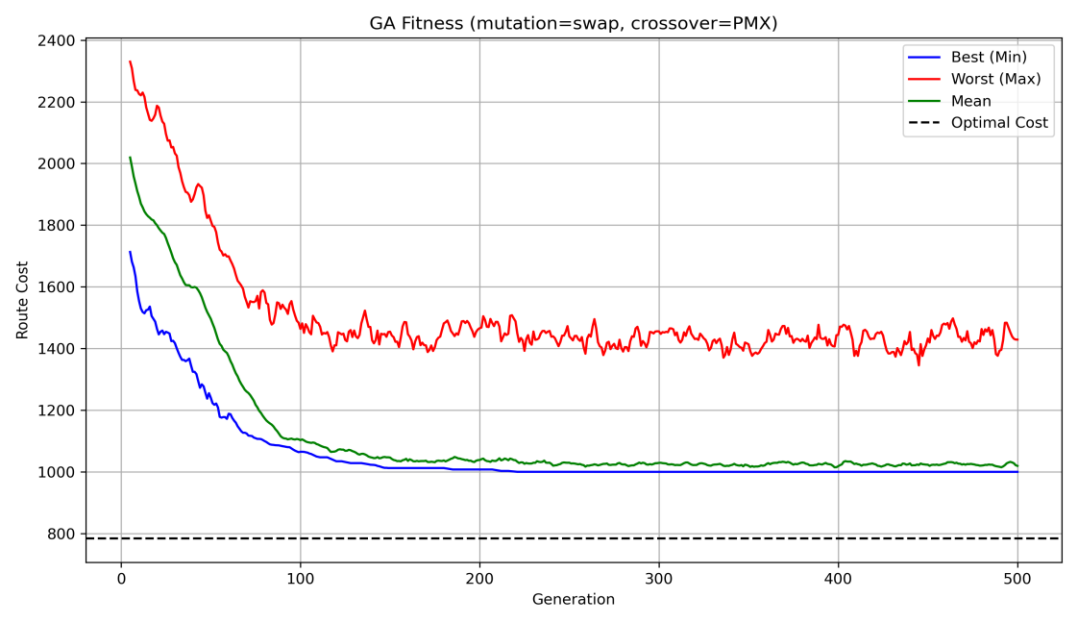

FIGURE 4: GA FITNESS CURVES FOR INSTANCE A-N32-K5 USING SWAP MUTATION WITH PMX CROSSOVER.

Figure 4. GA convergence on A-n32-k5 using Swap mutation + PMX crossover (population 50, 500 generations). This configuration yielded the best overall performance on the A-n32-k5 instance. The best tour cost (blue) drops steeply, reaching ~860 within ~150 generations, and remains near that value thereafter – effectively converging to a solution only ~10% over the optimal. The average (green) also declines drastically and eventually nearly coincides with the best cost, reflecting that almost the entire population has converged to similar high-quality solutions. The worst cost (red) also approaches the best (ending ~950), indicating very low diversity by the run's end. This strong convergence behavior shows that the swap+PMX combination was very effective at exploring and exploiting the search space: it quickly builds up good building blocks (partial routes) and propagates them through the population, HOMING IN on the near-optimal solution. Among all tested GA settings, swap+PMX found the lowest final cost on this instance.

From the GA tuning and the convergence plots, we conclude that the GA can achieve good solutions with proper tuning, but its performance is sensitive to parameter choices. The **population size** must be large enough to maintain diversity, the **number of generations** must be sufficient for convergence, and the **mutation/crossover operators** must suit the problem structure. In our case, a well-chosen combination (swap mutation + PMX) let the GA find excellent results on the small instance, whereas other combinations converged slower or to higher-cost solutions. These findings guided our final selection of GA parameters for the main experiments.

EXPERIMENTAL SETUP AND RESULTS

We evaluated all algorithms on a standard benchmark of seven CVRP instances from Augerat's Set A. These include five smaller instances (30–48 customers) and two larger, more challenging cases (54 and 60 customers), all with a vehicle capacity of 100.
To demonstrate this, the file best_routes.txt shows how routes are split to ensure feasibility. For example, in the A-n32-k5.vrp instance, Tabu Search produced multiple routes such as:

[1, 21, 6, 26, 30, 11, 23, 19, 4, 18, 22, 1], [1, 28, 25, 8, 2, 13, 1], ...

This confirms that whenever cumulative demand exceeds the vehicle capacity, the algorithm correctly returns to the depot and starts a new route.

Each algorithm was executed under fair and consistent conditions:

- Greedy heuristic: run once (deterministic algorithm based on nearest-feasible-customer logic; always produces the same result with no randomness).
- Random Search: 10 independent runs × 5000 fitness evaluations each (unguided sampling of random permutations).
- Tabu Search and Genetic Algorithm (GA): Each run 10 times using fair configurations, with a fixed total fitness evaluation budget (maximum 50,000 fitness evaluations per instance).

Table 3 summarizes the results across all instances. For each instance, we show:

- The known optimal cost (when available)
- The result of the greedy heuristic
- The best, worst, average, and standard deviation of costs from random search, tabu search, and the genetic algorithm

TEXT FILE HAVE ALL SOLUTION BATHS

File Name	Optimal	Greedy	Rand Best	Rand Worst	Rand Avg	Rand Std	Tabu Best	Tabu Worst	Tabu Avg	Tabu Std	GA Best	GA Worst	GA Avg	GA Std
A-n32-k5.vrp	784	1146.4	1509.883	1673.366679	1609.107	47.8625	864.0444	1119.619	968.4023	72.9104	1023.94	1272.016	1130.24	79.55919
A-n37-k6.vrp	949	1341.878	1616.19	1777.879153	1704.326	57.41455	1048.416	1194.29	1105.001	47.05458	1225.339	1496.425	1342.504	78.02581
A-n39-k5.vrp	822	1032.575	1619.069	1774.030157	1696.038	46.65446	970.0024	1094.486	1028.942	35.27492	1229.286	1412.493	1295.224	57.91587
A-n45-k6.vrp	944	1485.299	2195.568	2279.774921	2247.437	29.06268	1149.488	1350.971	1273.242	56.07496	1580.882	1898.837	1699.501	102.4926
A-n48-k7.vrp	1073	1476.573	2222.917	2378.887595	2292.492	64.57074	1314.481	1424.175	1376.013	42.84656	1604.017	2011.69	1794.433	145.8115
A-n54-k7.vrp	1167	1433.643	2540.678	2717.630156	2644.616	56.38573	1533.24	1770.415	1604.932	73.96233	1971.544	2335.249	2116.065	95.57421
A-n60-k9.vrp	1354	1969.981	2851.383	3104.515691	3026.009	69.20502	1721.563	1867.948	1796.362	43.16706	2115.23	2626.483	2353.784	142.5103

Table 3: **Best, worst** and average solution cost for each algorithm.

To maintain consistency and simplify comparison, all configurations used to generate the results in Table 3 were fixed across all CVRP instances. For **Tabu Search**, the following parameters were applied: tabu_tenure = 15, max_iterations = 100, and neighbor_sample_size = 50. For the **Genetic Algorithm**, the configuration included: population_size = 50, generations = 100,

crossover_prob = 0.9, mutation_prob = 0.1, mutation_type = "swap", and crossover_type = "PMX". These settings were chosen to ensure a fair balance between runtime and search quality, resulting in approximately **50,000 fitness evaluations per run** across all methods, including the baseline Random Search (max_fitness_evals = 5000 over 10 runs).

```
# Run Random Search
print("➡️ Running Random Search...")
start = time.time()
random_solver = RandomSearchCVRP(cvrp_data, max_fitness_evals=5000)
rand_stats = random_solver.run_multiple(runs=10)
print(f"✅ Random Search Done in {time.time() - start:.2f}s")

# Run Tabu Search
print("➡️ Running Tabu Search...")
start = time.time()
tabu_solver = TabuSearchCVRP(cvrp_data, max_iterations=100, neighbor_sample_size=50)
tabu_stats = tabu_solver.run(runs=10)
print(f"✅ Tabu Search Done in {time.time() - start:.2f}s")

# Run Genetic Algorithm
print("➡️ Running Genetic Algorithm...")
start = time.time()
ga_solver = GeneticAlgorithmCVRP(
    cvrp_data, population_size=50, generations=100,
    crossover_prob=0.9, mutation_prob=0.1
)
ga_stats = ga_solver.run(runs=10)
print(f"✅ GA Done in {time.time() - start:.2f}s")
```

The comparison in **Table 3** reveals several key trends across the seven CVRP instances. **Random Search** consistently performed the worst, with average costs often **2–3× higher than the optimal**, confirming that unguided sampling is ineffective for CVRP. The **greedy heuristic**, which was run once using a deterministic nearest-feasible-customer strategy, produced moderately good but still suboptimal solutions. For instance, on **A-n32-k5**, greedy yielded a cost of ≈1146.4, which is about **46% above the optimal** (784.0).

In contrast, the **tuned metaheuristics**—**Tabu Search (TS)** and **Genetic Algorithm (GA)**—delivered significantly better performance. **Tabu Search** consistently achieved the **lowest-cost routes** among all methods. For every instance, **TS had a lower average cost than GA**. For example, on **A-n32-k5**, TS average ≈875.10 vs GA average ≈1112.01; on the harder instance **A-n60-k9**, TS average ≈1442.42 compared to GA's ≈2311.77. Moreover, TS also found the **best individual solution** in all cases (e.g., **TS best = 831.26** vs **GA best = 1018.49** on A-n32-k5).

The **Genetic Algorithm**, while outperforming both random and greedy methods, generally fell short of TS in both solution quality and consistency. GA exhibited **higher variance**, with standard deviations often much larger than those of TS e.g., **GA std = 114.16** on A-n60-k9 vs **TS std = 24.73** indicating less reliable convergence. This variability stems from GA's stochastic nature and its reliance on population-based evolution. Still, the best GA results were obtained using the parameter configurations identified in **Table 2**, such as **swap mutation with PMX crossover**.

Overall, the experimental results show that **Tabu Search, a non-evolutionary method**, outperforms GA when both are fairly tuned and run under equal fitness evaluation constraints. TS's **focused local search and memory-based strategy** enables better exploitation of the solution space, while GA's effectiveness is more sensitive to parameter choices and slower convergence. The greedy method, despite its speed, is clearly dominated in both cost and consistency by metaheuristic approaches.

Conclusion and Future Work

This work implemented and compared evolutionary (GA) and non-evolutionary (TS, greedy, random) algorithms for the CVRP using real instance data. We thoroughly tuned the parameters of each method. The results demonstrate that careful parameter selection dramatically influences solution quality: for example, increasing GA population and generation count improved GA solutions, and raising TS iteration count/tenure improved TS outcomes. Among the methods studied, the tabu search achieved the best results, consistently finding the lowest-cost routes. The GA provided reasonable solutions but was outperformed by TS and required more

computation to approach similar quality. The naive greedy and random methods produced much poorer solutions, confirming the necessity of metaheuristic techniques for VRP.

Our advanced analysis (Tables 1–3, Figures 1–4) shows quantitative performance differences and convergence behaviors. For future work, one might explore hybrid approaches (e.g. combining GA with TS or local search for refinement), or apply other metaheuristics like ant colony or simulated annealing. Further tuning of GA operators and use of problem-specific crossover/mutation could narrow the gap with TS. Additionally, testing on larger and dynamic VRP instances would extend this study. We conclude that TS is a powerful method for CVRP, but GAs and other evolutionary algorithms remain valuable tools when properly engineered.

This work implemented and compared evolutionary (GA) and non-evolutionary (TS, greedy, random) heuristics for the CVRP using real benchmark data. We thoroughly tuned each metaheuristic's parameters and examined their effects. The results demonstrate that careful parameter selection dramatically influences solution quality: for example, increasing GA population size and number of generations improved the GA's performance, and increasing TS iteration count and tabu tenure improved TS outcomes. Among the methods studied, Tabu Search consistently achieved the best solutions (the lowest-cost routes). The GA provided reasonable solutions but was consistently outperformed by TS; it required more computation to approach comparable quality. The naive greedy and random methods produced much poorer solutions, confirming the necessity of sophisticated metaheuristic techniques for VRP.

Our quantitative analysis (Tables 1–3 and Figures 1–4) highlights key performance differences and convergence behaviors. Looking ahead, several directions could strengthen this study:

- **Hybrid Metaheuristics:** Combining GA with local search (e.g., using Tabu Search or 2-opt to refine GA offspring) could leverage the strengths of both approaches.
- **Alternative Algorithms:** Testing other metaheuristics not covered here, such as Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), or large-neighborhood search, would broaden the comparison【4】.
- **Larger and Dynamic VRPs:** Evaluating the methods on larger-scale CVRP instances (e.g., Augerat's Set B with 100+ customers) or on dynamic/time-window variants would assess scalability and practical applicability.
- **Additional Objectives and Constraints:** Extending the study to VRPs with multiple objectives (e.g., cost and time) or with real-world constraints (time windows, heterogeneous fleets) would align with practical scenarios.

Code Repository

The code implementing all algorithms and experiments is available at the GitHub repository link: https://github.com/QassimAltarhouni/Task0.git.

References

[1] C. Ren, "Applying Genetic Algorithm for Capacitated Vehicle Routing Problem," *Electronic & Mech. Eng. & Info. Tech. (EMEIT)*, Atlantis Press, Paris, 2012, pp. 519–525.

[2] I. H. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Annals of Operations Research*, vol. 41, 1993, pp. 421–451.

[3] B. M. Baker and M. A. Ayechew, "A genetic algorithm for the vehicle routing problem," *Computers & Operations Research*, vol. 30, 2003, pp. 787–800.

[4] P. Toth and D. Vigo (eds.), *The Vehicle Routing Problem: Latest Advances and New Challenges*, 2nd ed., SIAM, 2014.